Frequent Pattern Mining

- Mining Association Rules

Outline

- 何謂頻繁樣式的勘測 （frequent pattern mining）?
- 頻繁樣式的勘測方法
- 植基於條件式的（Constraint-based）頻繁樣式勘測
- 循序樣式（sequential patterns）
- 頻繁樣式的應用
- 頻繁樣式的研究問題

頻繁樣式的勘測

- 頻繁樣式(Frequent patterns): patterns (set of items, sequence, etc.)在資料庫中經常出現的樣式/模式 (pattern:項目集、順序等) [AIS93]
- 頻繁樣式的勘測: 找出資料中的規律(regularities)
  – What products were often purchased together? — Beer and diapers?!
  – What are the subsequent purchases after buying a PC?
  – What kinds of DNA are sensitive to this new drug?
  – Can we automatically classify web documents?

F.P.M. : frequent pattern mining

F.P.M. 是data mining的基本功能/工作

- 許多data mining task的基礎
  – Association rules, correlation, causality
  – sequential patterns, temporal/cyclic association, partial periodicity
  – spatial and multimedia patterns
  – associative classification
  – cluster analysis
  – iceberg cube, …
- 廣泛的應用
  – 購物籃分析
  – 交叉行銷
  – 型錄設計
  – 行銷活動分析
  – web log (click stream)分析, DNA sequence analysis, …

基本觀念:頻繁項目集

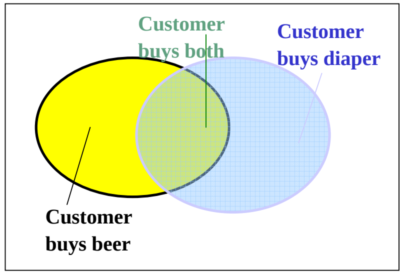
- 項目集 Itemset $X=\{x_1, \ldots, x_k\}$
  - 例：{A,C}, {B,E,F}, {C,E}
- 項目集的支持度(support)
  - s(A) = 3/4
- 頻繁項目集: 符合最小支持度（ m.s.: minimum support ）的項目集
- 勘測頻繁樣式:找出所有的頻繁樣式

Pattern = set of items

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F

基本觀念:關聯規則

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F



- 關聯規則的勘測:頻繁項目集找出後，決定「有興趣的」項目集之間的關係
  - 信賴度 confidence, *c*, 某交易如果包含 X ，此交易同時會包含Y的條件機率
  - 支持度 support, *s*,某交易包含 X∪Y的機率
- *m.s. = 50%, m.c. = 50%*
  - $A \rightarrow C$ (50%, 66.7%)
  - $C \rightarrow A$ (50%, 100%)

m.s.: minimum support 最小支持度
m.c.: minimum confidence 最小信賴度

探勘關聯規則（例）

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F

Min. support 50%
Min. confidence 50%

Frequent pattern	Support
{A}	75%
{B}	50%
{C}	50%
{A, C}	50%

rule $A \rightarrow C$:

support = support($\{A\}\cup\{C\}$) = 50%

confidence = support($\{A\}\cup\{C\}$)/support($\{A\}$) = 66.6%

關聯規則的類別

- 布林式的（boolean ）與數量式的（quantitative）
  - buys(x, “SQLServer”) ^ buys(x, “DMBook”) → buys(x, “DM Software”) [0.2%, 60%]
  - age(x, “30..39”) ^ income(x, “42..48K”) → buys(x, “PC”) [1%, 75%]
- 單一維度(single dimension)與多維度(multiple dimensional)
- 單一層次(single level)與多層次(multiple-level)
  - What brands of beers are associated with what brands of diapers?

關聯規則的延伸與應用

- Correlation, causality analysis & mining interesting rules
- Maxpatterns and frequent closed itemsets
- Constraint-based mining
- Sequential patterns
- Association-based classification
- Computing iceberg cubes

頻繁樣式的勘測方法

- Apriori 方法與其變化、改進
  - 不產生「候選樣式」的探勘方法
- 最大樣式(max-patterns)與封閉樣式(closed -patterns)
  - 精簡的表示方式
- 多維度、多層次頻繁樣式
- 有意義程度(Interestingness): correlation and causality

Apriori: Candidate Generation-and-test

- 產生「候選者」後檢視方式
  - 由長度 (k) 的「當選者」（frequent itemset) 產生長度 (k+1) 的「候選者」(candidate itemset)
  - 針對資料庫數其支持度，檢視是否是frequent
- 「候選者」條件：其子集合必定是frequent
  - 特性：anti-monotone
  - 包含 {beer, diaper, nuts}的交易必定包含 {beer, diaper}
  - 若{beer, diaper, nuts} frequent → {beer, diaper}一定 frequent
  - 任一 non-frequent 項目集之superset根本就不可能是頻繁項目集（所以就不會是「候選者」，不必產生，也不必數）
  - 可以排除許多「無用的組合」

Apriori 方法實例

Database

m.s. = 2

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1st scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

sup 以count表示，未以%表示

Apriori 演算法細節

L_1 = {frequent items}; k = 1 ;

if $L_k = \varnothing$ **stop** ;

C_{k+1} = 由 L_k *產生*;

對資料庫 D 中每一個交易 *t* 執行

所有包含於 *t* 的、C_{k+1}中的 candidate的個數 (support count) 加一

L_{k+1} = C_{k+1}之candidate滿足最小支持者 (minimum support)

$k = k + 1$;

答案： L_k*的聯集*;

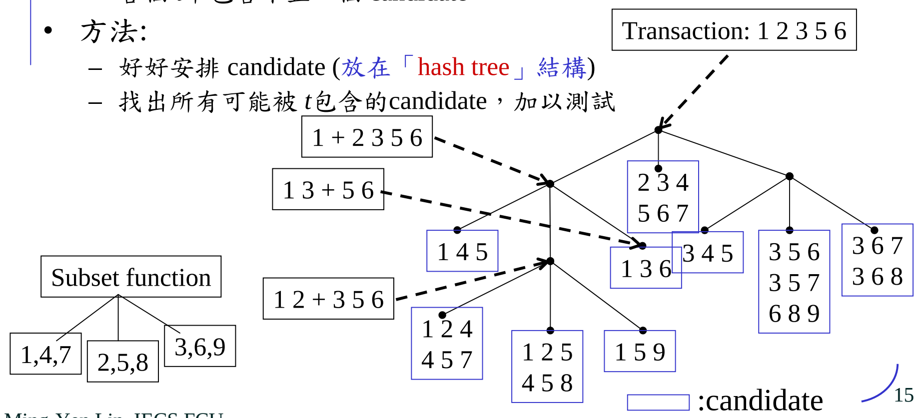
- L_k :大小為 k 的 frequent itemset
- C_k:大小為 k 的candidate itemset

Apriori 關鍵細節 I

- C_{k+1} = 由 L_k *產生*;
  - Step 1: self-joining L_k （ L_k 自交）
  - Step 2: pruning（消去不可能者）
- 例：L_3={*abc, abd, acd, ace, bcd*}依序排好
  - Self-joining: L_3*L_3
    - *abcd*： 由 *abc* and *abd*
    - *acde*：由 *acd* and *ace*
  - Pruning:
    - *消去 acde* :因 *ade* 不在 L_3
- C_4={*abcd*}

Apriori 關鍵細節 II

- 所有包含於 *t* 的 candidate的count加一; 難在哪？
  - candidate的總個數太多
  - 各個 *t*中包含不止一個 candidate
- 方法:
  - 好好安排 candidate (放在「hash tree」結構)
  - 找出所有可能被 *t*包含的candidate，加以測試



勘測方法的挑戰

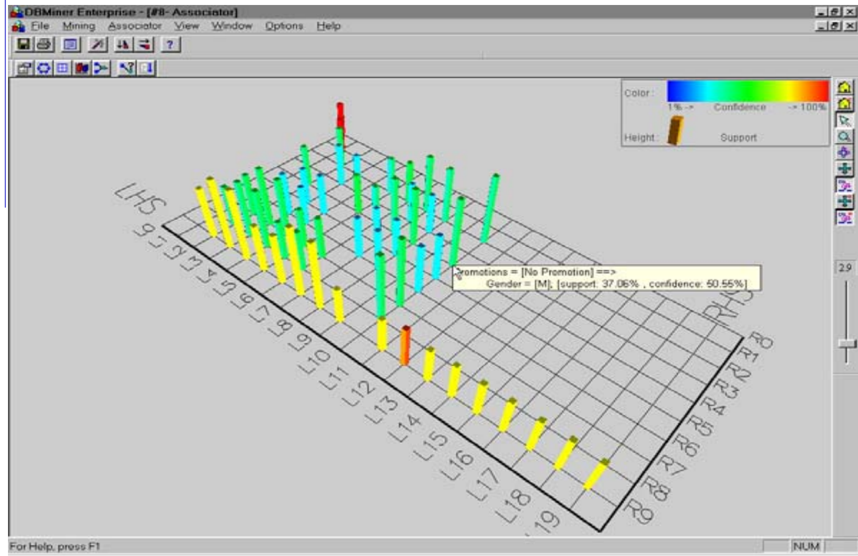
Challenges (也就是Apriori 方法變化改進的方向)

- 全部資料庫檢視次數太多
  - 想辦法減少
- candidates個數還是太多
  - 想辦法減少
- 數 candidate support 太麻煩
  - 想辦法數快點

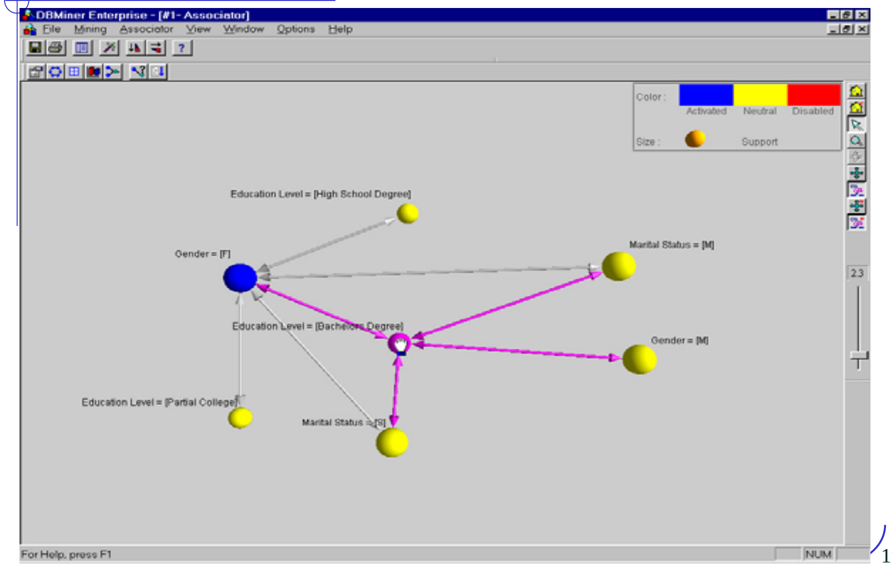
其他方法

- Apriori的改良
  - DIC
  - DHP
  - Partition
  - Sampling
  - …
- 不產生candidate，壓縮資料庫再找
  - FP-Growth, H-mine
- 用項目的交集（資料庫改以直向排列）
  - Eclat/MaxEclat ,VIPER

Association Rules 視覺化：: Pane Graph



Association Rules 視覺化：Rule Graph



最大樣式Max-patterns

- Frequent pattern $\{a_1, \ldots, a_{100}\}$ → $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100}-1 = 1.27*10^{30}$ frequent sub-patterns!
- 最大樣式 Max-pattern: 沒有「真」(proper) super pattern的樣式 (PS: frequent)
  - BCDE, ACD are max-patterns
  - BCD is not a max-pattern

m.s.=2

Tid	Items
10	A,B,C,D,E
20	B,C,D,E,
30	A,C,D,F

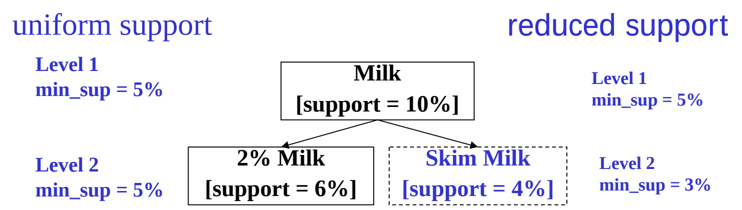
封閉樣式(Frequent Closed Patterns)

- Conf(ac→d)=100% ➔ 只紀錄 acd 就好
- 對於某一個頻繁項目集 X, 如果沒有項目 y 造成以下情形「每一個包含 X的交易也包含 y 」，則X 稱為一個封閉樣式
  - “acd” is a frequent closed pattern
- 頻繁樣式的一種精簡表示方式
- 簡化樣式與規則的個數

TID	Items
10	a, c, d, e, f
20	a, b, e
30	c, e, f
40	a, c, d, f
50	c, e, f

多層次Association Rules

- 項目通常有其概念架構
- 設定 support 要具彈性: 低層次的 support應該比較低
- 依據維度與層次將交易資料庫編碼
- 探索多層次的探勘



多維度Association

- 單一維度（dimension）規則

  buys(X, “milk”) ⇒ buys(X, “bread”)
- 多維度: ≥ 2 維度或陳述（predicate）
  - 維度內(Inter-dimension) assoc. rules (*no repeated predicates*)

    age(X,”19-25”) ∧ occupation(X,“student”) ⇒ buys(X,“coke”)
  - hybrid-dimension assoc. rules (*repeated predicates*)

    age(X,”19-25”) ∧ buys(X, “popcorn”) ⇒ buys(X, “coke”)
- 類別屬性(Categorical Attributes)
  - finite number of possible values, no ordering among values
- 數量屬性(Quantitative Attributes)
  - numeric, implicit ordering among values

Distance-based Association Rules

- Binning 不見得抓得住區間資料的語意(semantic)

Price($)	Equi-width (width $10)	Equi-depth (depth 2)	Distance-based
7	[0,10]	[7,20]	[7,7]
20	[11,20]	[22,50]	[20,22]
22	[21,30]	[51,53]	[50,53]
50	[31,40]		
51	[41,50]		
53	[51,60]		

- 以距離為主的分割更距離散化(discretization)考量
  - density/number of points in an interval
  - “closeness” of points in an interval

有意義程度Interestingness Measure: Correlations

- *play basketball* ⇒ *eat cereal* [40%, 66.7%] 誤導！
  - The overall percentage of students eating cereal is 75% which is higher than 66.7%.
- *play basketball* ⇒ *not eat cereal* [20%, 33.3%] 更精準, 雖然 support and confidence 較低
- 度量相依性/相關事件: lift

$$corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}$$

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

植基於條件式的頻繁樣式勘測

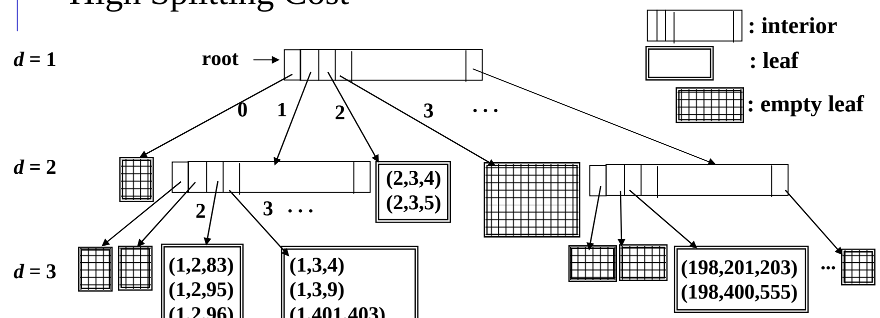
- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!
- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface)
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
  - System optimization: explores such constraints for efficient mining—constraint-based mining

LexMiner

- Ming-Yen Lin and Suh-Yin Lee, "A Fast Lexicographic Algorithm for Association Rule Mining in Web Applications," Proceedings of the ICDCS Workshop on Knowledge Discovery and Data Mining in the World-Wide Web (ICDCS00), Taipei, Taiwan, R.O.C., pp. F7-F14, 2000.

Apriori use Hash Tree to stored Candidates

- Excess Comparison, Eg. ***T*$_1$ = {1, 2, 3, 4, 5, 6}**
- Duplicate Counting Avoidance, Eg. ***T*$_2$ = {1, 3, 4, 200, 401, 403}**
- Large Storage Requirement
- High Splitting Cost

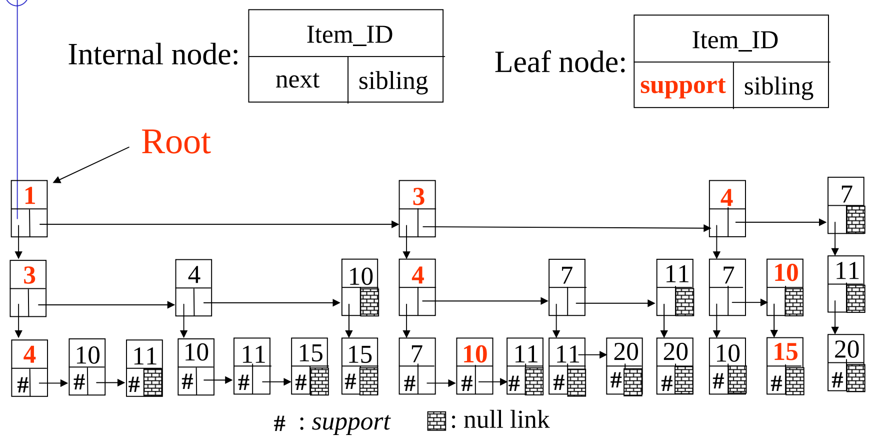


hash function = *x* MOD *199*, size of each leaf = 5, *d* = depth

LexTree: **Lex**icographically Ordered **Tree**

- Intrinsic Property: Lexicographic Order
  - Items in each transaction, eg. {*7*, *11*, *20*, *29*, *37*}
  - *k*-itemsets, eg. (*1*, *3*, *4*) < (*1*, <u>*3*</u>, *10*) < (*1*, <u>*4*</u>, *10*)
- Storing **itemsets**: by lexicographic order
- LexTree: compact, hierarchical tree
  - candidate LexTree: **efficient**, **redundant-free** <u>support counting</u>
  - frequent LexTree: effective <u>candidate generation</u>
- LexMiner Algorithm

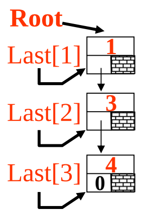
LexTree Structure



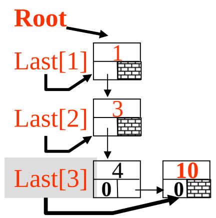
(***1***, ***3***, ***4***), (*1*, *3*, *10*), (*1*, *3*, *11*), (*1*, *4*, *10*), (*1*, *4*, *11*), (*1*, *4*, *15*), (*1*, *10*, *15*), (*3*, *4*, *7*), (***3***, ***4***, ***10***), (*3*, *4*, *11*), (*3*, *7*, *11*), (*3*, *7*, *20*), (*3*, *11*, *20*), (*4*, *7*, *10*), (***4***, ***10***, ***15***), (*7*, *11*, *20*)

LexTree Construction

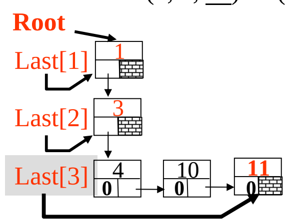
(1) Insert candidate (***1***, ***3***, ***4***)



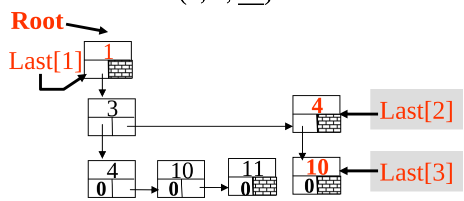
(2) Insert candidate (***1***, ***3***, <u>***10***</u>)



(3) Insert candidate (***1***, ***3***, <u>***11***</u>)



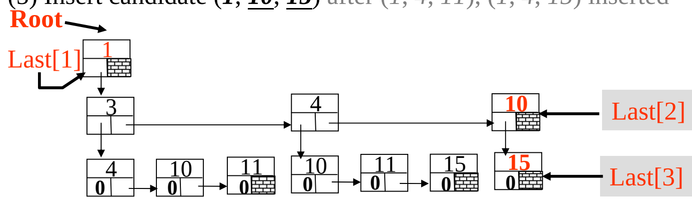
(4) Insert candidate (***1***, ***4***, <u>***10***</u>)



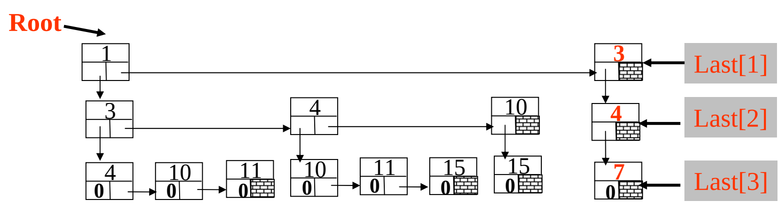
Insert <u>candidates</u> (***1***, ***3***, ***4***), (***1***, ***3***, <u>***10***</u>), (***1***, ***3***, <u>***11***</u>), (***1***, <u>***4***</u>, ***10***)

LexTree Construction (Cont.)

(5) Insert candidate (***1***, <u>***10***</u>, <u>***15***</u>) after (*1*, *4*, *11*), (*1*, *4*, *15*) inserted



(6) Insert candidate (<u>***3***</u>, <u>***4***</u>, <u>***7***</u>)

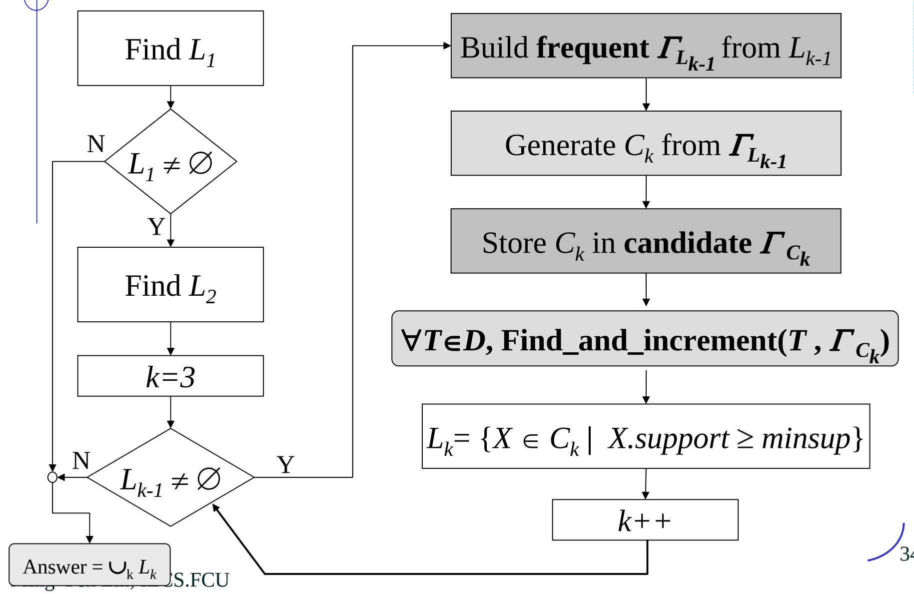


Notations

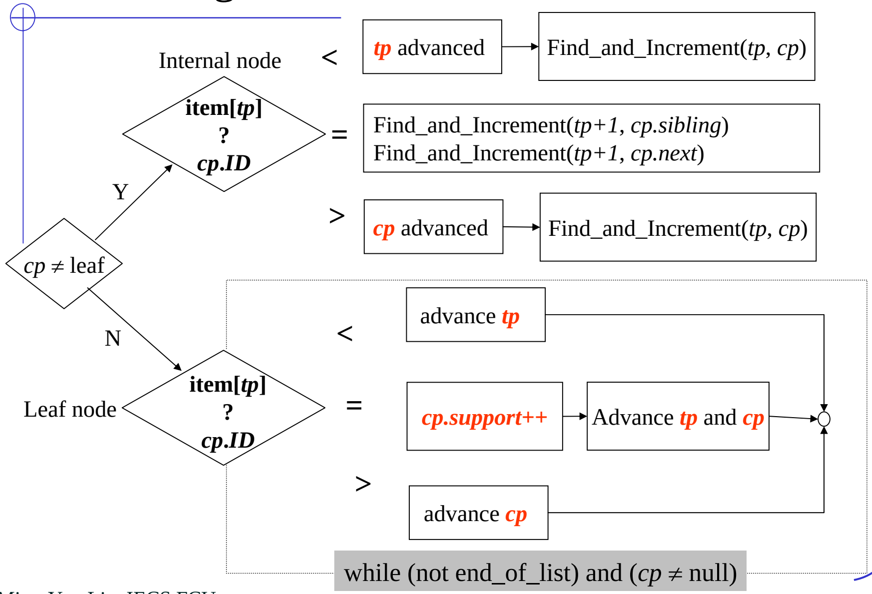
- D : The database of transactions
- T : A transaction, $T=\{x_1, x_2, \ldots, x_p, \ldots, x_m\}$
  $x_1, x_2, \ldots, x_k$: Items
- *minsup* : The minimum support specified by the user
- X : k-itemset, $X=(x_1, x_2, \ldots, x_k)$
- *X.support* : The support of itemset X

- C_k : The set of candidate k-itemsets
- L_k : The set of frequent k-itemsets
- Γ_{C_k} : The candidate k-itemset *LexTree*
- Γ_{L_k} : The frequent k-itemset *LexTree*

Algorithm LexMiner



Algorithm Find_and_Increment



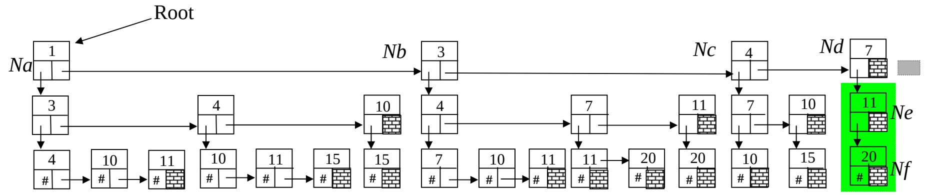
Example 1: #Comparison Minimized

- At pass 3, $T_1=\{7, 11, 20, 29, 37\}$

Intrinsically ordered 3-itemsets in T_1

$\{7, 11\} \times \{20, 29, 37\}$
$\{7, 20\} \times \{29, 37\}$
$\{7, 29\} \times \{37\}$
$\{11, 20\} \times \{29, 37\}$
$\{11, 29\} \times \{37\}$
$\{20, 29\} \times \{37\}$

$7 \times 11 \times (20, 29, 37)$
$\times 20 \times (29, 37)$
$\times 29 \times (37)$
$11 \times 20 \times (29, 37)$
$\times 29 \times (37)$
$20 \times 29 \times (37)$



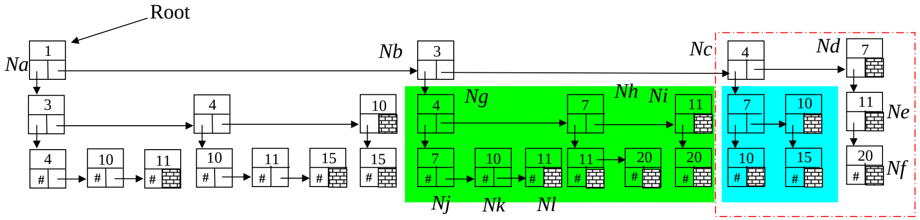
Candidate 3-itemset *LexTree*, Γ_{C_3}

Example 2: Fast Support Counting

- At pass 3, T_2={3, 4, 7, 10, 11}

Intrinsically ordered 3-itemsets in T_2

3 x 4 x (7,10,11)
x 7 x (10,11)
x 10 x (11)
4 x 7 x (10,11)
x 10 x (11)
7 x 10 x (11)



Candidate 3-itemset *LexTree*, Γ_{C_3}

Efficient Candidate Generation

- Common prefixed L_{k-1} : linked by sibling
- Join: $C_k = L_{k-1} \times L_{k-1}$, then

insert into C_k
select $p[1], p[2], \ldots, p[k-1], q[k-1]$
from L_{k-1} p, L_{k-1} q
where $p[1]=q[1], \ldots, p[k-2] = q[k-2], p[k-1] < q[k-1]$;

- Prune: candidate itemset having any subset that is not in $\Gamma_{L_{k-1}}$
  - Searching in $\Gamma_{L_{k-1}}$: similar technique for find_and_increment

FP-growth

- J. Han, J. Pei, and Y. Yin: "Mining frequent patterns without candidate generation". In Proc. ACM-SIGMOD'2000, pp. 1-12, Dallas, TX, May 2000.

- Compress DB into a tree (FP-tree)
- Find frequent itemsets in FP-tree

Construction of FP-tree from a Transaction Database

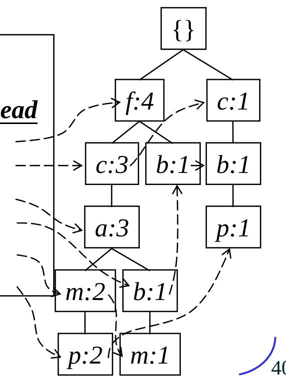
TID	*Items bought*	*(ordered) frequent items*
100	***{f, a, c, d, g, i, m, p}***	***{f, c, a, m, p}***
200	***{a, b, c, f, l, m, o}***	***{f, c, a, b, m}***
300	***{b, f, h, j, o, w}***	***{f, b}***
400	***{b, c, k, s, p}***	***{c, b, p}***
500	***{a, f, c, e, l, p, m, n}***	***{f, c, a, m, p}***

min_support = 0.5

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Order frequent items in frequency descending order
3. Scan DB again, construct FP-tree

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



Mining Frequent Patterns with FP-trees

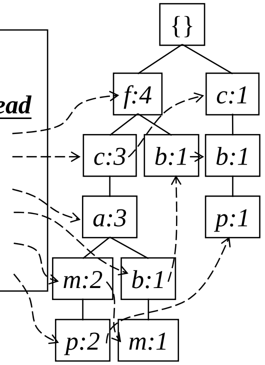
- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path —single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

From FP-tree to Conditional Pattern-Base

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item *p*
- Accumulate all of *transformed prefix paths* of item *p* to form *p*'s conditional pattern base

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



***Conditional* pattern bases**

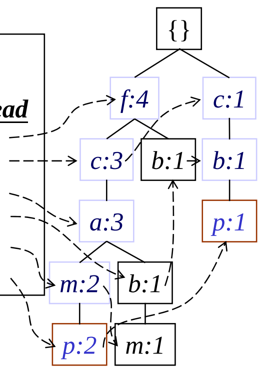
item	*cond. pattern base*
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

Transformed Prefix Paths

- Derive the *transformed prefix paths* of item *p*
  - For each item *p* in the tree, collect *p*'s prefix path with count = *p*'s frequency
  - Why only prefix path? Why this count? Complete?

Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



***Conditional* pattern bases**

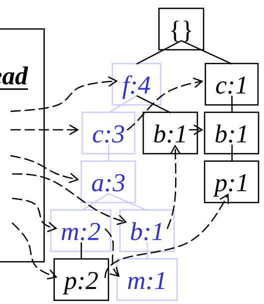
item	*cond. pattern base*
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

From Conditional Pattern-Bases to Conditional FP-trees

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base

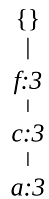
Header Table

Item	*frequency*	*head*
f	*4*	
c	*4*	
a	*3*	
b	*3*	
m	*3*	
p	*3*	



***m-conditional* pattern base:**
fca:2, fcab:1

➔



***m-conditional* FP-tree**

➔

All frequent patterns relate to *m*
m,
fm, cm, am,
fcm, fam, cam,
fcam

Recursion: Mining Each Conditional FP-tree Until …

{}
|
f:3
|
c:3
|
a:3

m-conditional **FP-tree**

Cond. pattern base of “am”: (fc:3)

{}
|
f:3
|
c:3

am-conditional **FP-tree**

Cond. pattern base of “cm”: (f:3)

{}
|
f:3

cm-conditional **FP-tree**

Cond. pattern base of “cam”: (f:3)

{}
|
f:3

cam-conditional **FP-tree**

A Special Case: Single FP-tree Path

- Suppose a (conditional) FP-tree T has a single path P
- The complete set of frequent patterns of T can be generated by enumeration of all the combinations of the sub-paths of P

{}
|
f:3
|
c:3
|
a:3

➔

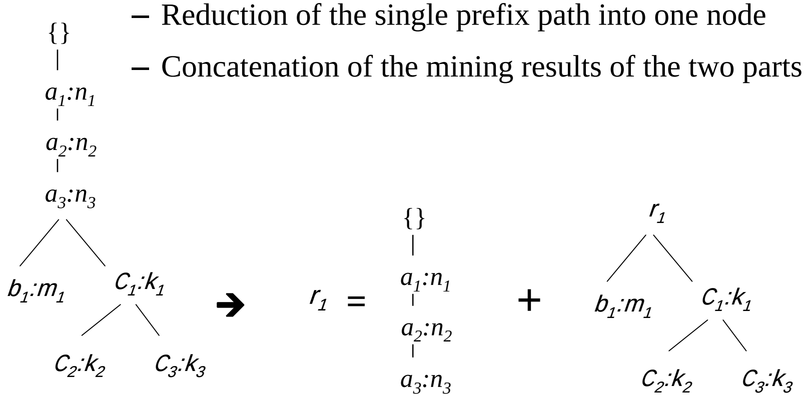
All frequent patterns concerning *m*

m,
fm, cm, am,
fcm, fam, cam,
fcam

m-conditional **FP-tree**

A More General (Special) Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts



循序樣式（sequential patterns）

- 交易資料庫 (transaction databases), 時序資料庫 (time-series databases) 與 序列資料庫 (sequence databases)
- 頻繁樣式(frequent patterns)與循序樣式(sequential patterns)
- 循序樣式的應用 Applications of sequential pattern mining
  - 顧客購買序列
    - First buy computer, then CD-ROM, and then digital camera, within 3 months.
  - 醫療處方, 天災 (e.g., earthquakes), 科學、工程程序, 股票等
  - 電話通聯樣式，Weblog click streams
  - DNA sequences and gene structures

何謂循序樣式之勘測?

- 給一堆序列(sequence)，找出所有 *frequent* subsequences（子序列）

A *sequence* : < (ef) (ab) (df) c b >

A *sequence database*

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

An element may contain a set of items. Items within an element are unordered and we list them alphabetically.

<a(bc)dc> is a *subsequence* of <a(abc)(ac)d(cf)>

Given *support threshold* *min_sup* =2, <(ab)c> is a *sequential pattern*

循序樣式勘測的困難何在？

- 隱藏於資料庫中的可能的循序樣式個數相當龐大可能
- 探勘演算法必須
  - 找出所有滿足 minimum support (frequency) threshold的樣式
  - 要具有高度效率與可擴充性，減少資料庫檢視次數
  - 可以跟各種使用者所指定的constraints 搭配

Sequential Patterns基本特性： Apriori

- 基本特性: Apriori (Agrawal & Sirkant'94)
  - If a sequence S is not frequent
  - then none of the super-sequences of S is frequent
  - 例如, <hb> infrequent → <hab> and <(ah)b> 也 infrequent

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Given *support threshold* *min_sup* =2

GSP 演算法細節

L_1 = {frequent sequence of length 1}; $k = 1$;

if $L_k = \varnothing$ **stop** ;

C_{k+1} = 由 L_k *產生*;

對資料庫 D 中每一個交易 *t* 執行

所有包含於 *t* 的、C_{k+1}中的 candidate的個數 (support count) 加一

$L_{k+1} = C_{k+1}$之candidate滿足最小支持者 (minimum support)

$k = k + 1$;

答案： L_k*的聯集*;

- L_k :大小為 *k* 的 frequent sequence
- C_k:大小為 *k* 的candidate sequence

找 Length-1的 Sequential Patterns

- Initial candidates: all singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan database once, count support for candidates

min_sup =2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Cand	Sup
<a>	**3**
<b>	**5**
<c>	**4**
<d>	**3**
<e>	**3**
<f>	**2**
~~<g>~~	1
~~<h>~~	1

產生 Length-2 Candidates

51 length-2 Candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

Without Apriori property, 8*8+8*7/2=92 candidates

Apriori prunes 44.57% candidates

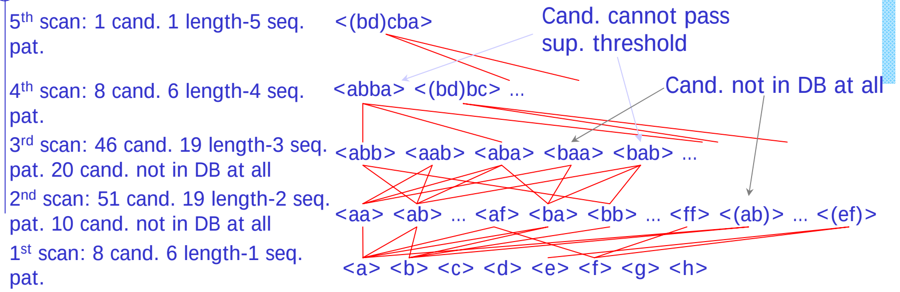
找Length-2 Sequential Patterns

- Scan database one more time, collect support count for each length-2 candidate
- There are 19 length-2 candidates which pass the minimum support threshold
  - They are length-2 sequential patterns

產生& 找出 Length-2 Candidates

- 產生
  - length-2 sequential patterns 自交 (Self-join)
    - Based on the Apriori property
    - <ab>, <aa> and <ba> are all length-2 sequential patterns → <aba> is a length-3 candidate
    - <(bd)>, <bb> and <db> are all length-2 sequential patterns → <(bd)b> is a length-3 candidate
  - 46 candidates are generated, prune the impossible
- 找 Length-3 Sequential Patterns
  - Scan database once more, collect support counts for candidates
  - 19 out of 46 candidates pass support threshold

The GSP Mining Process



min_sup =2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

GSP瓶頸

- candidates 個數太多
  - 1,000 frequent length-1 sequences: generate length-2

$$1000 \times 1000 + \frac{1000 \times 999}{2} = 1{,}499{,}500$$

- 全部資料庫檢視次數太多
- 真正難的：mining long sequential patterns
  - An exponential number of short candidates 天文數字
  - A length-100 sequential pattern needs 10^{30} candidate sequences!

$$\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}$$

頻繁樣式的應用

- 以關聯性來分類Association-based classification
- Iceberg cube computation
- Database compression by fascicles and frequent patterns
- Mining sequential patterns (GSP, PrefixSpan, SPADE, etc.)
- Mining partial periodicity, cyclic associations, etc.
- Mining frequent structures, trends, etc

頻繁樣式的研究問題

- Multi-dimensional gradient analysis: patterns regarding changes and differences
- Mining fault-tolerant associations
  - “3 out of 4 courses excellent” leads to A in data mining
- Fascicles and database compression by frequent pattern mining
- Partial periodic patterns
- DNA sequence analysis and pattern classification

Tools: Association Rule Mining

- Free
  - ARTool, http://www.cs.umb.edu/~laur/ARtool/
  - Apriori, http://fuzzy.cs.uni-magdeburg.de/~borgelt/#Software
- Commercial
  - IBM Intelligent Miner for Data, http://www.software.ibm.com/data/intelli-mine/
  - DBMiner 2.0, http://www.dbminer.com/
  - clementine, http://www.spss.com/clementine/

Apriori (1/2)

- apriori [options] infile outfile [appfile]
  - infile: file to read transactions from
  - outfile: file to write association rules
  - appfile: file stating item appearances (optional)
- options:

  -t# target type (s: item sets, r: rules (default), h: hyperedges)

  -m# minimal number of items per set/rule/hyperedge (default: 1)

  -n# maximal number of items per set/rule/hyperedge (default: 5)

  -s# minimal support of a set/rule/hyperedge (default: 10%)

  -c# minimal confidence of a rule/hyperedge (default: 80%)

Apriori (2/2) options

-b/f/r# blank characters, field and record separators (default: " \t\r", " \t", "\n")

-o use original definition of the support of a rule (body & head)

-p# output format for support/confidence (default: "%.1f%%")

-x extended support output (print both rule support types)

-a print absolute support (number of transactions)

-e# additional rule evaluation measure (default: none)

(# always means a number, a letter, or a string that specifies the parameter of the option.)

Apriori Input Format

- text file (field and record separators and blanks)
  - Record separators: lines
  - field separators fields (or columns): words
  - Blanks : fill fields (columns), e.g. to align them.
- Examples

```
1,2,3
1,4,5
2,3,4
1,2,3,4
2,3
1,2,4
4,5
1,2,3,4
3,4,5
1,2,3
```

Item Appearances File

- item may appear only in rule bodies (antecedents):
  – i in b body a ante antecedent
- item may appear only in rule heads (consequents):
  – o out h head c cons consequent
- item may appear in rule bodies (antecedents) or in rule heads (consequents):
  – io inout bh b&h ac a&c both
- item may appear neither in rule bodies (antecedents) nor in rule heads (consequents):
  – n neither none ign ignore -
- Example 1: Generate only rules with item "x" in the consequent.
  in
  x out

Sample Command

- apriori test1.tab test.rul
- apriori -b"(" -f, -r")" test2.tab test2.rul

(0,1)(0,2)(0,3)(1,1)(1,4)(1,5)(2,2)(2,3)(2,4)(3,1)(3,2)(3,3)(3,4)(4,2)(4,3)...

- apriori -f ",.;:" -l test3.tab test3.rul
- apriori test1.tab -

Example 2: Item "x" may appear only in a rule head (consequent),
item "y" only in a rule body (antecedent);
appearance of all other items is not restricted.
both
x head
y body

1,2,3
1,4,5
2.3.4
1,2,3,4
2:3
1,2,4
4,5
1,2,3,4
3;4;5
1,2,3

0 1
0 2
0 3
1 1
1 4
1 5
2 2
2 3
2 4
3 1
3 2
3 3
3 4
4 2
4 3
5 1
5 2
5 4
6 4
6 5
7 1
7 2
7 3
7 4
8 3
8 4
8 5
9 1
9 2
9 3

Sample Output

3 <- 2 (70.0%, 85.7%)
2 <- 3 (70.0%, 85.7%)
2 <- 1 (60.0%, 83.3%)
4 <- 5 (30.0%, 100.0%)
3 <- 2 1 (50.0%, 80.0%)
2 <- 3 1 (40.0%, 100.0%)
4 <- 3 5 (10.0%, 100.0%)
4 <- 1 5 (10.0%, 100.0%)
2 <- 3 4 1 (20.0%, 100.0%)

Applications 回顧

- 購物籃分析
- 交叉行銷
- 型錄設計
- 行銷活動分析
- web log (click stream)分析
- DNA sequence analysis
- ...